Extendable Minecraft Server Manager Documentation

Release 5.0.6b0

Benedikt Schmitt

Jul 12, 2017

Contents

CHAPTER 1

How to

Installation

1. Update the system packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

2. Install the depencies:

```
$ sudo apt-get install python3 python3-pip screen openjdk-7-jre-headless
```

   Note, that the EMSM needs at least **Python 3.2** to run.

3. Install the EMSM Python package from PyPi:

```
$ sudo pip3 install --pre emsm
```

   This will also install all EMSM Python depencies.

4. Create the user, that should run the EMSM:

```
$ sudo addgroup --system --no-create-home --disabled-login --group minecraft
$ sudo adduser --system --no-create-home --disabled-login --ingroup minecraft␣
↪minecraft
```

5. Create the instance folder. This folder will later contain all worlds and server executables:

```
$ sudo mkdir /opt/minecraft
```

6. Create the `/opt/minecraft/minecraft.py` EMSM launcher and add it to the global PATH:

```
#!/usr/bin/env python3

#/opt/minecraft/minecraft.py
```

```
import emsm

# Make sure, the instance dir is correct.
emsm.run(instance_dir="/opt/minecraft")
```

```
$ sudo chmod +x /opt/minecraft/minecraft.py
$ sudo ln -s /opt/minecraft/minecraft.py /usr/bin/minecraft
```

7. Make sure the `/opt/minecraft/` directory is owned by the minecraft user:

```
$ sudo chown -R minecraft:minecraft /opt/minecraft
```

8. Execute the EMSM:

```
$ sudo minecraft emsm --version
```

9. That's it. Your instance directory should now look like this:

```
|- /opt/minecraft
    |- conf
    |- logs
    |- minecraft.py
    |- plugins
    |- plugins_data
    |- server
    |- worlds
```

You probably want to use some plugins like the *guard*, *initd* or *backups* plugin. So don't forget to take a look at their documentation later.

Troubleshooting

WrongUserError

If you run the application under another user than *minecraft*, you have to edit the `conf/main.conf` configuration file before you call the EMSM the first time otherwise you will get a `WrongUserError`:

```
[emsm]
user = foobar
```

Configuration

The `conf/` directory contains all configuration files.

main.conf

The `main.conf` file contains the configuration of the EMSM and the plugins.

```
[emsm]

# User that should run all of your minecraft worlds.
user = minecraft

# Maximum time that is waited until another EMSM instance releases
# the file lock.
# A negative values means no timeout and wait endless if necessairy.
timeout = -1

# You can provide a *screenrc* file. Please note, that it must be an
# **absolute** path.
# This option is optional.
#
#screenrc = /opt/minecraft/conf/screenrc
screenrc =
```

Each plugin has its own section. E.g.:

```
[backups]
archive_format = bztar
restore_message = This world is about to be resetted to an earlier state.
restore_delay = 5
max_storage_size = 30
exclude_paths = logs
    mods
```

Some plugins allow you to override global options for each world. Please take a look at the documentation of the *Plugins* for further information.

server.conf

The `server.conf` allows you to adjust some properties of the internal EMSM server wrapper classes. Usually, it should not be necessairy to edit this configuration file, but some times you have to.

Examples

- You want to adjust the java heap size:

  ```
  [vanilla 1.8]
  # You can use these placeholders in the start_command:
  # * {server_exe}
  # * {server_dir}
  start_command = java -Xmx3G -jar {server_exe}
  ```

- You want to use the latest server version, but the EMSM contains an old url:

  ```
  [vanilla 1.8]
  url = https://...
  ```

  Make sure to update the server after changing the configuration:

  ```
  $ minecraft -s "vanilla 1.8" server --update
  ```

You can override some options for each world, like the *start_command*. This can be used to grant different worlds different amounts of memory. You will learn how to do this in the next section.

*.world.conf

Note: This is only the EMSM configuration for the world. You still have to edit the `server.properties` file in the world's directory.

Each world managed by the EMSM has its own configuration `.world.conf` file in `conf/`. We will now add the world *morpheus*:

```
$ # In the conf/ directory:
$ touch morpheus.world.conf
```

This file is empty at the moment. On the next run of the EMSM, it will detect the configuration file and fill it with default values:

```
$ minecraft -W worlds --status
```

When you look into `morpheus.world.conf`, you can find the *world* section:

```
[world]
stop_timeout = 10
stop_message = The world is going to be stopped.
stop_delay = 10
server = vanilla 1.11
```

- **stop_timeout**

  The maximum time, waited until the world stopped after sending the `stop` command.

- **stop_message**

  This message is printed before sending the stop command to the world.

- **stop_delay**

  The time between the sending the *stop_message* and the *stop* command. If **stop_delay** and **stop_timeout** are both `10`, the stop takes at least 10 seconds and at maximum 20.

- **server**

  The name of the minecraft server that should power this world.

  Run `minecraft server --list` to get a list of all supported minecraft server. If your server is not listed, you can create a new plugin, which provides a *server wrapper*.

You can overridde some global plugin and server options for each world:

```
[server:vanilla 1.11]
start_command = java -Xmx1G -jar {server_exe} nogui

[plugin:backups]
max_storage_size = 10
exclude_paths = logs
    mods
```

The configuration section for a server is the server name, prefixed with `server:` and the section for a plugin is the plugin's name, prefixed with `plugin:`.

Please note, that you only overridde the configuration for a *specific* server, not the current server of the world:

```
# Has no effect, because the world is configured to use "vanilla 1.11",
# and not "bungeecord".
[server:bungeecord]
start_command = echo "Hallo"
```

Check out the *Plugins* documentation, if you want to know more about their configuration.

Example

```
# This configuration file contains the configuration for the world
#
#     **morpheus**
#
# This file can be used to override global configuration values in
# the *server.conf* and *emsm.conf* configuration files.
#
# [world]
# stop_timeout = int
# stop_message = string
# stop_delay = int
# server = a server in server.conf
#
# Custom options for the backups plugin:
#
# [plugin:backups]
# archive_format = bztar
# max_storage_size = 30
#
# Custom options for the vanilla 1.8 server:
#
# [server:vanilla 1.8]
# start_command = java -Xms512m -Xmx1G -jar {server_exe} nogui
#

[world]
stop_timeout = 10
stop_delay = 5
stop_message = The server is going down.
    Hope to see you soon.
server = vanilla 1.11

[plugin:backups]
max_storage_size = 10
archive_format = zip
exclude_paths = logs
    mods
    crash-reports

[plugin:initd]
enable = yes
```

First steps

There are some common arguments and run types you should know:

- The **help** argument:

```
$ minecraft -h
$ minecraft worlds -h
$ minecraft server -h
$ minecraft backups -h
...
```

- The **long-help** argument:

```
$ minecraft worlds --long-help
$ minecraft backups --long-help
...
```

Each plugin provides its own arguments, similar to *git*. There are only a few **global arguments** to unify the interface:

- Select all worlds:

```
$ minecraft -W [plugin ...]
$ minecraft --all-worlds [plugin ...]
```

- Select world by world:

```
$ minecraft -w foo -w bar [plugin ...]
$ minecraft --world foo --world bar [plugin ...]
```

- Select all server software:

```
$ minecraft -S [plugin ...]
$ minecraft --all-server [plugin ...]
```

- Select server by server:

```
$ minecraft -s vanilla -s bukkit [plugin ...]
$ minecraft --server vanilla --server bukkit [plugin ...]
```

Common tasks

- Start all worlds:

```
$ minecraft -W worlds --start
$ minecraft --all-worlds worlds --start
```

Note: Please note, that the *first start* of a world may fail, if the *eula* has not been accepted.

- Restart one world:

```
$ minecraft -w foo worlds --restart
$ minecraft --world foo worlds --restart
$ minecraft -w foo worlds --force-restart
```

- Stop all worlds:

```
$ minecraft -W worlds --stop
$ minecraft --all-worlds worlds --stop
```

- Server update:

```
$ minecraft -S server --update
$ minecraft -s "vanilla 1.8" server --update
$ minecraft --server "vanilla 1.8" server --update
```

Updates

From time to time, the EMSM receives some updates. Especially the server database and the server download urls. So how can you update the EMSM?

Server updates

The server software is usually updated faster than the EMSM database. But don't worry, you can often use the latest server software with the EMSM.

Let's assume, the *minecraft server 1.8* received a patch from mojang and you want to use it:

1. Edit the `server.conf` configuration file:

```
[vanilla 1.8]
# Setting the url here, will overwrite the value in the EMSM database.
url = https://s3.amazonaws.com/Minecraft.Download/versions/1.8.1/minecraft_
↪server.1.8.1.jar
```

1. Update the server with the *server* plugin:

```
$ minecraft -s "vanilla 1.8" server --update
```

If the update fails, the old server software will be restored and nothing changed.

Please take a look at the *server configuration* and the *server* plugin for more information.

EMSM updates

The EMSM is a Python package and you can simply update it using *pip*:

```
$ pip3 install --upgrade emsm
```

Since the instance folder is not touched by this command, there is no need for a backup before an update anymore.

If the EMSM does not work as expected after the update, take a look at the *Changelog* or create an issue.

Upgrade

If the major version number changes, you should take a look at the *Changelog* first. There will be an upgrade guide and additional information.

This is a quick installation guide. I guess it will not take more than **15 minutes** to set the application up and learn how it works.

CHAPTER 2

Plugins

emsm.plugins.backups

About

Extends the EMSM by a backup manager.

Download

You can find the latest version of this plugin in the EMSM GitHub repository.

Configuration

main.conf

```
[backups]
archive_format = bztar
restore_message = This world is about to be ressetted to an earlier state.
restore_delay = 5
max_storage_size = 30
backup_logs = yes
exclude_paths =
```

archive_format

Is the name of the archive format used to create the backups. This string has to be listed in *shutil.get_archive_formats()*. Usually, there should be at least *zip* or *tar* available.

restore_message

Is send to the world's chat before restoring the world.

restore_delay

Seconds between sending the *restore_message* to the chat and starting the restore.

max_storage_size

Maximum number of backups in the storage folder, before older backups will be removed.

backup_logs

If `yes`, the log files are included into the backup, otherwise not.

exclude_paths

If given, these `glob()` like paths in a world folder are not included into the backup.

Use one line for a path.

See also:

`shutil.ignore_patterns()`

*.world.conf

Some global configuration options can be overridden for each world:

- *archive_format*
- *max_storage_size*
- *backup_logs*
- *exclude_paths*

```
# In a *.world.conf configuration file
[plugin:backups]
max_storage_size = 10
exclude_paths = logs
    banned-ips.json
    crash-reports
```

Arguments

Note: All arguments will only affect the worlds selected with *–world* or *–all-world*

`--list`
Lists all available backups.

`--create`
Creates a new backup.

`--restore` `PATH`
Restores the world with the backup from the given BACKUP_PATH.

`--restore-latest`
Restores, if available, the latest backup of the world.

`--restore-menu`
Opens a menu, where the user can select which backup he wants to restore.

Cron

You should create a cronjob to create daily backups:

```
# m h dom mon dow user command
# Creates a backup of all worlds everyday at 2:00h
0 2 *   *   *   root minecraft -W backups --create
```

Backup archive structure

A typical backup archive has this structure:

```
o
|- world_conf.json         # The EMSM configuration of the world
|- world                   # the minecraft world
    |- server.log
    |- server.properties
    |- ...
```

Changelog

EMSM v3

- changed package structure and dropped support for EMSM v2 backups.

EMSM v5

- the world's dedicated configuration *file* is now saved, instead of the world's configuration *section*. This also means, that we can not restore the configuration of backups created with EMSM v3. The worlds can still be restored.

emsm.plugins.emsm

About

Provides information about the EMSM itself, like the version and simplifies the EMSM update.

Download

You can find the latest version of this plugin in the EMSM GitHub repository.

Arguments

--version
: Shows the current EMSM version number.

--license
: Shows the EMSM license.

Changelog

- EMSM 5.0.3b0

  Removed the **–check-update** action. Using *pip* to check for updates is more reliable and the preferred way:

```
$ pip3 list -o | grep emsm
```

  - https://github.com/benediktschmitt/emsm/issues/67
  - https://github.com/benediktschmitt/emsm/issues/69

emsm.plugins.guard

About

Monitors selected worlds (*–world*, *-w*, *-W*) and reacts on issues.

Download

You can find the latest version of this plugin in the EMSM GitHub repository.

Configuration

Since EMSM version 3.2.2-beta, this plugin requires no more configuration. The command line arguments allow you to adjust the guard for each world.

Arguments

When invoked all worlds selected with the global EMSM commands *-W* or *-w* are checked.

--error-action {none, **stop**, **restart**}
Defines how the guard handles a world in trouble.

Note: Per default, all tests will be performed. If you don't want to run all tests, you can pass the tests, which should be performed as command line arguments.

--test-status
Check if the world is online.

--test-log
Check if the logs contain an error.

--test-port
Check if the world is reachable.

--output-format {console, **text**}
Defines the output format.

text is suitable for sending the guard output via email.

--output-only-new-warnings
If an error with the world has been detected in the previous run, the warning for this world will be suppressed.

Cron

This plugin is made for cron (therefore it does not print much):

```
# m h dom mon dow user command
# Runs the guard every 5 minutes for all worlds
*/5 * *   *   *   root minecraft -W guard --output-only-new-warnings --output-format␣
↪text

# Runs the guard every 5 minutes for the world *foo*.
*/5 * *   *   *   root minecraft -w foo guard --output-only-new-warnings --output-
↪format text
```

Changelog

3.0.0-beta

- Removed configuration options that were dedicated to enable the guard for selected worlds.
- The *new* guard simply monitors all worlds selected with the **-W** or **-w** argument.

3.2.2-beta

- removed configuration options
- added port check again
- added different output formats

emsm.plugins.hellodolly

About

This plugins works as a tutorial. It's inspired by the wordpress plugin Hello Dolly.

Code and Download

You can find the latest version of *hello_dolly* on the EMSM GitHub GitHub repository.

```
#!/usr/bin/env python3

# The MIT License (MIT)
#
# Copyright (c) 2014-2016 Benedikt Schmitt <benedikt@benediktschmitt.de>
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
```

```
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.

"""
About
-----

This plugins works as a tutorial. It's inspired by the wordpress plugin
`Hello Dolly <https://wordpress.org/plugins/hello-dolly/>`_.

Code and Download
-----------------

You can find the latest version of *hello_dolly* on the EMSM GitHub
`GitHub repository <https://github.com/benediktschmitt/emsm/blob/master/plugins/
↪plugins.py>`_.

..
    By the way, this is a comment block in reST.

    The next line is a little hack. Unfortunetly, sphinx does not find
    "hellodolly.py" when this docstring is included with autodoc. So this does
    not work:

    .. literalinclude:: hellodolly.py

    The next line is actually a small hack. When the documentation is
    built, this module is included from
    ``EMSM_ROOT/docs/source/plugins/``, but the module is in
    ``EMSM_ROOT/plugins/```.

.. literalinclude:: ../../../emsm/plugins/hellodolly.py

Installation
------------

We want to distribute our plugin because we think it brings so much joy
to all players. So let's create a small package.

This is usually done with the :mod:`plugins.plugins` plugin:

.. code-block:: bash

    $ foo@bar: ls
    hellodolly.py ...
    $ foo@bar: plugin.py --source hellodolly.py
    $ foo@bar: ls
    hellodolly.py hellodolly.tar.bz2 ...

The compressed package archive should now be in your working directory.

Usage
-----
```

```
.. code-block:: bash

    $ foo@bar: # Will print only one row:
    $ foo@bar: minecraft -W hellodolly

    $ foo@bar: # Prints 5 rows or less, if the configuration value is smaller:
    $ foo@bar: minecraft -W hellodolly --rows 5

Documentation
-------------

Acutally, EMSM uses sphinx *autodoc* feature to create the documentation for
the plugins. So what, you see here is the docstring of the ``hellodolly.py``
module.
"""


# Modules
# ------------------------------------------------

# std
import os
import random

# third party
import termcolor

# local
import emsm
from emsm.core.base_plugin import BasePlugin


# Data
# ------------------------------------------------

# This variable helps the EMSM to find the actual plugin class in this module.
PLUGIN = "HelloDolly"

# These are the well-known hello dolly lyrics.
LYRICS = """Hello, Dolly
Well, hello, Dolly
It's so nice to have you back where you belong
You're lookin' swell, Dolly
I can tell, Dolly
You're still glowin', you're still crowin'
You're still goin' strong
We feel the room swayin'
While the band's playin'
One of your old favourite songs from way back when
So, take her wrap, fellas
Find her an empty lap, fellas
Dolly'll never go away again
Hello, Dolly
Well, hello, Dolly
It's so nice to have you back where you belong
You're lookin' swell, Dolly
I can tell, Dolly
```

```
You're still glowin', you're still crowin'
You're still goin' strong
We feel the room swayin'
While the band's playin'
One of your old favourite songs from way back when
Golly, gee, fellas
Find her a vacant knee, fellas
Dolly'll never go away
Dolly'll never go away
Dolly'll never go away again"""


# Classes
# ------------------------------------------------

class HelloDolly(BasePlugin):

    # We don't need to wait for other plugins, so we don't care
    # about the init priority. If you want that your plugin is initialised
    # earlier than others, make this value smaller.
    INIT_PRIORITY = 0

    # Also, we don't care about if the finish method of our plugin is called
    # early or late. The *finish* method of plugins with a smaller
    # *FINISH_PRIORITY* is called earlier.
    FINISH_PRIORITY = 0

    # At the moment, there is no direct url to the latest version of this
    # plugin.
    # In the future, the plugin manager could use this url to detect new
    # versions of your plugin and will download them automatically.
    DOWNLOAD_URL = None

    # The last compatible version of the EMSM.
    VERSION = "5.0.0-beta"

    # The EMSM automatically uses the DESCRIPTION variable to set up the
    # *--long-help* argument parser argument.
    #
    # We ususally use here the module's docstring. Note, that ``__doc__``
    # does not interfere with the HelloDolly docstring ``HelloDolly.__doc__``
    # since the HelloDolly class has no docstring.
    DESCRIPTION = __doc__

    def __init__(self, application, name):
        """
        """
        # We need to init the BasePlugin. This is necessairy, so that we can
        # safely access:
        #
        #   * self.global_conf()
        #   * self.argparser()
        #   * ...
        BasePlugin.__init__(self, application, name)

        # The configuration and argument parser are set up in own methods
        # for readability.
        self._setup_conf()
```

```
        self._setup_argparser()
        return None

    def _setup_conf(self):
        """
        Sets the global configuration up. (The ``hellodolly`` section in
        :file:`main.conf`)
        """
        # Get the configuration dictionary for this plugin.
        conf = self.global_conf()

        # This is an example of the hellodolly configuration section in the
        # main.conf configuration file:
        #
        # [hellodolly]
        # max_rows = 5
        #

        self._max_rows = conf.getint("max_rows", 5)
        conf["max_rows"] = str(self._max_rows)
        return None

    def _setup_argparser(self):
        """
        Sets the argument parser up.
        """
        # Get the plugin's argument parser.
        parser = self.argparser()

        parser.description = (
            "Demonstrates the implementation of a plugin. Inspired by the "
            "wordpress plugin \"Hello, Dolly\"."
            )
        parser.epilog = "https://github.com/benediktschmitt/emsm"

        # Note, that we prefix the *dest* value, since all arguments share
        # the same namespace.
        parser.add_argument(
            "--rows", "-r",
            action = "store",
            dest = "hellodolly_rows",
            type = int,
            default = 1,
            metavar = "ROWS",
            help = "The number of lines that will be printed."
            )
        return None

    def _uninstall(self):
        """
        If you created data not stored in ``data_dir()`` or used also the
        *worlds.conf* or *server.conf* configuration files, you should ask the
        user here, if he wants to remove these files and settings too.

        Note the difference between ``_uninstall()`` and ``uninstall()``.
        """
        # Your uninstallation stuff here
        # ...
```

```
        return None

    def run(self, args):
        """
        Writes lines of our lyrics into the chats of the selected worlds.

        Parameters:
            * args
                Is a namespace that contains the parsed arguements.
        """
        # Get the number of lines we want to print and make sure, that
        # the number is not greater then the max_rows configuration value.
        rows = args.hellodolly_rows
        if rows > self._max_rows:
            rows = self._max_rows
        if rows < 0:
            rows = 0

        # Run hellodolly for each world, which has been selected with
        # *-w* or *-W* per command line.
        # We sort the worlds by their names, to process them in alphabetical
        # order.
        worlds = self.app().worlds().get_selected()
        worlds.sort(key = lambda w: w.name())

        for world in worlds:
            self.be_poetic(world, rows)
        return None

    def get_lyrics(self, num_rows):
        """
        Returns rows of the hello dolly lyrics.

        Parameters:
            * num_rows
                The number of rows, that should be extracted from the
                lyrics.
        """
        global LYRICS
        lyrics = LYRICS

        # Get *num_rows* lines of the lyrics.
        lyrics = lyrics.split("\n")
        if num_rows > len(lyrics):
            return lyrics
        else:
            a = random.randint(0, len(lyrics) - num_rows)
            lyrics = lyrics[a:a+num_rows]
        return lyrics

    def be_poetic(self, world, num_rows):
        """
        Writes the *lyrics* to the chat of all running, selected worlds.
        """
        lyrics = self.get_lyrics(num_rows)

        # We follow the inofficial EMSM style guide and print the
        # world name in cyan.
```

```
        print(termcolor.colored("{}:".format(world.name()), "cyan"))
        if world.is_offline():
            print("\t", termcolor.colored("error:", "red"), "world is offline")
        else:
            for row in lyrics:
                world.send_command("say {}".format(row))
            print("\t", "world has been visited")
        return None

    def finish(self):
        """
        This method is always called, when the EMSM is about to finish.
        It should be used for clean up or background stuff.
        """
        return None
```

Installation

We want to distribute our plugin because we think it brings so much joy to all players. So let's create a small package.

This is usually done with the `plugins.plugins` plugin:

```
$ foo@bar: ls
hellodolly.py ...
$ foo@bar: plugin.py --source hellodolly.py
$ foo@bar: ls
hellodolly.py hellodolly.tar.bz2 ...
```

The compressed package archive should now be in your working directory.

Usage

```
$ foo@bar: # Will print only one row:
$ foo@bar: minecraft -W hellodolly

$ foo@bar: # Prints 5 rows or less, if the configuration value is smaller:
$ foo@bar: minecraft -W hellodolly --rows 5
```

Documentation

Acutally, EMSM uses sphinx *autodoc* feature to create the documentation for the plugins. So what, you see here is the docstring of the `hellodolly.py` module.

emsm.plugins.initd

About

Works as interface between the linux *initd service* and the EMSM.

Download

You can find the latest version of this plugin in the EMSM GitHub repository.

Installation

You can use this plugin with *initd* or *systemd*.

initd (`/etc/init.d/`)

You only have to create the `init.d` script `/etc/init.d/minecraft`:

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          EMSM - extendable minecraft server manager
# Required-Start:    $remote_fs $syslog $network
# Required-Stop:     $remote_fs $syslog $network
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts and stops your minecraft worlds.
### END INIT INFO

EMSM=`which minecraft`
PLUGIN=initd

test -x $EMSM || exit 0

case "$1" in
   start)
      $EMSM $PLUGIN --start
      ;;
   stop)
      $EMSM $PLUGIN --stop
      ;;
   restart)
      $EMSM $PLUGIN --restart
      ;;
   status)
      $EMSM $PLUGIN --status
      ;;
   *)
      echo "Usage: $0 {start|stop|restart|status}" >&2
      exit 1
      ;;
esac
```

```
$ sudo chmod +x /etc/init.d/minecraft
$ sudo update-rc.d minecraft defaults
$ sudo update-rc.d minecraft enable
```

systemd (`/etc/systemd/system/minecraft.service`)

You only have to create the `/etc/systemd/system/minecraft.service` file:

```
[Unit]
Description=Extendable Minecraft Server Manager (EMSM)
Requires=network.target
After=network.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/bin/minecraft initd --start
ExecStop=/usr/bin/minecraft initd --stop
ExecReload=/usr/bin/minecraft initd --restart

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable minecraft.service
```

Configuration

*.worlds.conf

```
[plugin:initd]
enable = yes
```

enable

If `yes`, the autostart/-stop is enabled.

Arguments

`--start`
Starts all worlds, where initd has been enabled in the `*.world.conf` configuration file.

`--stop`
Stops all worlds, where initd is enabled. Note, that this will always **force** the stop of the world, since the process is killed anyway during system shutdown.

`--restart`
Forces the restart of all worlds, for which initd has been enabled.

`--status`
Prints the status (online/offline) for each *initd* enabled world.

Exit code

The exit code is set to:

- 0 if no error occured.
- 2 if an error occured.

Changelog

EMSM v5

- *initd* must now be enabled in the *plugin:initd* configuration section of the `*.world.conf` configuration file.

  In v4 (`worlds.conf`):

  ```
  [morpheus]
  enable_initd = yes
  ```

  In v5 (`morpheus.world.conf`):

  ```
  [plugin:initd]
  enable = yes
  ```

emsm.plugins.plugins

About

This is a package manager for EMSM plugins. Uninstall and install plugins with this plugin.

This plugin works only with valid packages and plugins that store its data in the dedicated paths.

Download

You can find the latest version of this plugin in the EMSM GitHub repository.

Arguments

--install ARCHIVE
: Installs an new plugin from the archive. If a plugin with the same name already exists, the installation will fail.

--remove PLUGIN
: Removes the plugin from the EMSM. Please make sure, that no other plugin depends on this one.

--list
: Lists all loaded plugins.

Package structure

The archive that contains the plugin should have the following structure:

```
|- foo.tar.bz2
   |- plugin.py
   |- data
      |- bar.txt
      |- bar.csv
      |- ...
```

During the installation, the path names will be changed to:

```
|- EMSM_ROOT
  |- plugins
     |- foo.py     <= plugin.py
  |-plugins_data
     |- foo        <= data
        |- bar.txt
        |- bar.csv
        |- ...
```

Builder

This plugin comes with an EMSM independent building script for new plugins. This means, that you can call this script without having the EMSM environment.

Arguments

--create TARGET

--source FILE

--data DIRECTORY

--help, -h

Example

Build the plugin *foo*, that comes with a data directory:

```
$ plugin.py --create build/foo --source dev/foo.py --data dev/foo_data
$ ls build
... foo.tar.bz2 ...
```

emsm.plugins.server

About

This plugin provides a user interface for the server wrapper. It can handle the server files and their configuration parameters easily.

Download

You can find the latest version of this plugin in the **EMSM** GitHub repository.

Configuration

```
[server]
update_message = The server is going down for an update.
    Come back soon.
```

update_message

Message sent to a world before stopping the world due to an server update.

Arguments

Note: Make sure to select the server via `-s, --server`.

`--usage`
Prints the names of the worlds, powered by a server.

`--list`
Prints the names of all server supported by the EMSM.

`--update`
Updates the server software.

emsm.plugins.worlds

About

This plugin provides a user interface for the server wrapper. It handles the server files and their configuration parameters easily.

Download

You can find the latest version of this plugin in the **EMSM** GitHub repository.

Configuration

```
[worlds]
default_log_start = 0
default_log_limit = 10
open_console_delay = 1
send_command_timeout = 10
```

default_log_start

Is the first line of the log, that is printed. Can be overwritten by a command line argument.

default_log_limit

Is the default number of log lines, that is printed at once. This value can be overwritten by a command line argument too.

open_console_delay

Time between printing the WARNING and opening the console.

send_command_timeout

Maximum time waited for the response of the minecraft server, if the `--verbose-send` command is used.

Arguments

--address
Prints the binding (ip, port) of the world.

--configuration
Prints the section of the world in the `worlds.conf`.

--directory
Prints the directory path that contains the world.

--log
Prints the log.

--log-start `LINE`
The first line of the log that is printed. If *'-10'* (with quotes!), the 10th last line will be the first line that is printed.

--log-limit `LINES`
Limits the number of printed lines.

--pid
Prints the PID of the screen session that runs the server.

--status
Prints the status of the world (online or offline).

--send `CMD`
Sends the command to the world.

Note: Escaping commands with **spaces**

If you want to send a command like `say Hello players!`, you have to escape it.

```
minecraft -W worlds --send 'say Hello players!'
```

--verbose-send `CMD`
Sends the command to the server and prints the echo in the logfiles.

--console
Opens the server console.

--start
Starts the world

--stop

Warning: Stopping the world not using the dedicated commands, will **not** call the **event dispatcher** and may cause bugs.

Stops the world

--force-stop
Like –stop, but kill the processes if the world is still online after the smooth stop.

--kill

> **Warning:** Using this command can cause data loss.

Kills the process of the world.

--restart
Restarts the world. If the world is offline, the world will be started.

--force-restart
Like –restart, but forces the stop of the world if necessairy.

--uninstall
Removes the world and its configuration.

Examples

```
# Start all worlds:
$ minecraft -W worlds --start

# Send a command to the server and print the console output:
$ minecraft -W worlds --verbose-send list
$ minecraft -W worlds --verbose-send '"say Use more TNT!"'

# Print the log of the world *foo*:
$ minecraft -w foo worlds --log
$ minecraft -w foo worlds --log-start '-20'
$ minecraft -w foo worlds --log-limit 5
$ minecraft -w foo worlds --log-start '-50' --log-limit 10

# Open the console of a running world
$ minecraft -w bar worlds --console

...
```

What are EMSM plugins?

The EMSM plugins work as *frontend* for the EMSM and automate minecraft server tasks.

How to write a plugin

It's very easy to create your own plugin.

- Read the *hellodolly* tutorial for a quick introduction.
- If you want to know more and you're fit in Python, I suggest you read the source code of the EMSM for a full overview. I guess this will not take more than 1h of your time. Simply start with the `__init__.py` module and follow the calls.

CHAPTER 3

API

emsm.core.application

exception emsm.core.application.**ApplicationException**
Bases: Exception

Base class for all exceptions in this module.

exception emsm.core.application.**WrongUserError**(*required_user*)
Bases: *emsm.core.application.ApplicationException*

Raised if the EMSM is executed by the wrong user.

class emsm.core.application.**Application**(*instance_dir*)
Bases: object

This class manages the initialisation and the complete run process of the EMSM.

An EMSM application should be executed in a code structure similar to this one:

```
app = Application()
try:
    app.setup()
    app.run()
except Exception as err:
    app.handle_exception()
    raise
finally:
    exit(app.finish())
```

argparser()
Returns the EMSM *ArgumentParser*, that is used internally.

conf()
Returns the used *Configuration* instance.

exit_code()
Returns the exit code of the application.

See also:

set_exit_code()

finish()
Performs some clean up and background stuff.

Returns *exit_code()*

Note: Do not mix this method up with the *emsm.core.plugins.PluginManager.finish()* method. These are not related.

See also:

- *run()*
- *exit_code()*

handle_exception()
Checks `sys.exc_info()` if there is currently an uncaught exception and logs it.

paths()
Returns the used *Pathsystem* instance.

plugins()
Returns the used *PluginManager* instance.

run()
Runs the plugins.

See also:

- *emsm.core.plugins.PluginManager.run()*
- *emsm.core.plugins.PluginManager.finish()*

server()
Returns the used *ServerManager* instance.

set_exit_code(*code*)
Sets the exit code to *code*. This is the exit code, that is used for the Python `exit()` function.

Raises

- **TypeError** – if *code* is not an int.
- **ValueError** – if *code* < 0.

See also:

- *exit_code()*
- `exit()`

setup()
Initialises all components of the EMSM.

This method will block, until the EMSM filelock could be acquired or the configuration timeout value is reached.

worlds()
Returns the used *WorldManager* instance.

emsm.core.argparse_

This module contains the *ArgumentParser* class which wraps a Python argparse.ArgumentParser for the EMSM.

class emsm.core.argparse_.**LongHelpAction**(*option_strings*, *description=None*, *dest='==SUPPRESS=='*, *default='==SUPPRESS=='*)
Bases: argparse.Action

Prints the *description* using less (if available) and exists.

class emsm.core.argparse_.**ArgumentParser**(*app*)
Bases: object

Wraps an argparse.ArgumentParser instance.

This class handles the *root* EMSM argument parser. The root parser only has a few global EMSM commands like `-w`, `-s`. Each plugin has its own subparser:

```
$ foo@bar: minecraft [emsm args] (plugin_name) [plugin args]
```

Example:

```
# Call the *worlds* plugin with the world *foo* as target.
$ foo@bar: minecraft -w foo worlds --status
```

argparser()
Returns the wrapped argparse.ArgumentParser instance.

args(*cache=True*)
Parses (if not yet done) the command line arguments and returns a namespace object that contains the result.

Parameters **cache** (*bool*) – If `True`, and the arguments have already been parsed, the result of the previous parse is returned.

See also:

- argparse.ArgumentParser.parse_args()

plugin_parser(*plugin_name*)
Returns the subparser for the plugin with the name *plugin_name*.

setup()
Adds the global EMSM arguments to the root argument parser.

This method has to be called, when the *WorldManager* and *ServerManager* have been loaded, since we require the names of the available worlds and server.

emsm.core.base_plugin

class emsm.core.base_plugin.**BasePlugin**(*app*, *name*)
Bases: object

This is the base class for all plugins.

If you want to know, how to implement your own plugin, you should also take a look at the plugins.hellodolly plugin.

DESCRIPTION = ''
This string is displayed when the --long-help argument is used.

DOWNLOAD_URL = ''
The plugin package can be downloaded from this url.

See also:

- *emsm.plugins.plugins* package manager

FINISH_PRIORITY = 0
Integer with the finish priority of the plugin. A higher value results in a later call of the finish method.

HIDDEN = False
If True, the plugin has no *argparser()* and can therefore not be invoked from the command line.

INIT_PRIORITY = 0
Integer with the init priority of the plugin. A higher value results in a later initialisation.

VERSION = '0.0.0'
The last version number of the EMSM version that worked correctly with that plugin.

app()
Returns the parent EMSM Application that owns this plugin.

argparser()
Returns the argparse.ArgumentParser that is used by this plugin.

If *HIDDEN* is True, *None* is returned, since the plugin has no argument parser.

See also:

- *emsm.core.argparse_.ArgumentParser.plugin_parser()*

conf()
Returns a dictionary like object that contains the configuration of the plugin.

Deprecated since version 4.0.16-beta: This method has been replaced by *global_conf()* to clarify the difference to *world_conf()*.

data_dir(*create=True*)
Returns the directory that contains all data created by the plugin to manage its EMSM job.

Parameters **create** (*bool*) – If the directory does not exist, it will be created.

See also:

- emsm.core.paths.Pathsystem.plugin_data_dir()

finish()
Called when the EMSM application is about to finish. This method can be used for background jobs or clean up stuff.

Subclass:

- You may override this method.

See also:

- *emsm.core.plugins.PluginManager.finish()*

global_conf()
Returns a dictionary like object, that contains the *global* configuration of the plugin (`plugins.conf`).

Seealso *world_conf()*

name()
Returns the name of the plugin.

plugin_uninstalled = <blinker.base.NamedSignal object at 0x7fe9e9d57b00; 'plugin_uninstalled'>
Signal, that is emitted, when a plugin has been uninstalled.

run(*args*)
The *main* method of the plugin. This method is called if the plugin has been invoked by the command line arguments.

Params argparse.Namespace args is an argparse.Namespace instance that contains the values of the parsed command line arguments.

Subclass:

- You may override this method.

See also:

- *argparser()*
- *emsm.core.argparse_.ArgumentParser.args()*
- *emsm.core.plugins.PluginManager.run()*

uninstall()
Called when the plugin should be uninstalled. This method is interactive and requires the user to confirm if and which data should be removed.

The BasePlugin removes:

- the plugin module (the *.py* file in *plugins*)
- the plugin data directory
- the plugin configuration

Subclass:

Subclasses should override the `_uninstall()` method.

Signals:

- *plugin_uninstalled*

See also:

- *data_dir()*

- *conf()*
- _uninstall()

world_conf(*world*)
Returns a dictionary like object, that contains the *world* specific configuration of the plugin (`foo.world.conf`).

Seealso *global_conf()*

Parameters **world** – The `WorldWrapper` of the world or the world's name (str).

emsm.core.conf

class emsm.core.conf.**ConfigParser**(*path*)
Bases: `configparser.ConfigParser`

Extends the standard Python `configparser.ConfigParser` by some useful methods.

Parameters **path** (*str*) – The path to the configuration file. This file is used, when you call *read()* or *write()*.

epilog()
Returns a comment, which is written at the begin of a configuration file.

path()
Returns the path of the configuration file.

read()
Reads the configuration from *path()*.

remove()
Removes the configuration file from the file system.

write()
Writes the configuration into *path()*.

class emsm.core.conf.**MainConfiguration**(*path*)
Bases: *emsm.core.conf.ConfigParser*

Handles the `main.conf` configuration file.

This file includes the configuration for the EMSM Application and the plugins.

The EMSM owns the `[emsm]` section and each plugin has its own section with the plugin name.

```
[emsm]
user = minecraft
timeout = 0
screenrc =

[backups]
include_server = ...
# ...
```

epilog()

class emsm.core.conf.**ServerConfiguration**(*path*)
Bases: *emsm.core.conf.ConfigParser*

Handles the *server.conf* configuration file, which allows the user to overwrite the default EMSM settings for a server wrapper like the *url* or the *start command*.

See also:

- *emsm.core.server.BaseServerWrapper.conf()*
- *emsm.core.conf.WorldConfiguration()*

epilog()

class emsm.core.conf.**WorldConfiguration**(*path*)
Bases: *emsm.core.conf.ConfigParser*

Handles a configuration file for *one* world and allows the user to set custom configuration values for each plugin, server and the EMSM.

Parameters **path** (*str*) –

epilog()

class emsm.core.conf.**Configuration**(*app*)
Bases: object

Manages all configuration files of an EMSM application object.

See also:

- *emsm.core.application.Application.conf()*
- emsm.core.paths.Pathsystem.conf_dir()

list_worlds()
Returns a list with the names of all worlds, for which a configuration file has been found.

main()
Returns the *MainConfiguration*.

read()
Reads all configration files.

server()
Returns the *ServerConfiguration*.

world(*name*)
Returns the *WorldConfiguration* for the world with the name *name* and None, if there is not such a world.

worlds()
Returns a list with all *WorldConfiguration* objects.

write()
Saves all configuration values.

emsm.core.license_

emsm.core.license_.**LICENSE** = **'The MIT License (MIT)\n\nCopyright (c) 2014-2016 Benedikt Schmitt <benedikt@bene**
The EMSM license

emsm.core.logging_

class emsm.core.logging_.**Logger**(*app*)
Bases: object

Sets the *root* logging.Logger up.

The EMSM logger queues all log records until the emsm.log can be acquired without side effects. This is the case, when the *Application* acquired the *file lock*. The queued records are then pushed to the emsm.log.

The EMSM logging stategy requires, that each module uses its own logger instance:

```
>>> import logging
>>> log = logging.getLogger(__file__)
```

setup()
Opens the emsm.log and pushes all queued log recods to the log file.

Hint: This method requires that the Application aquired the file lock.

emsm.core.paths

class emsm.core.paths.**Pathsystem**(*instance_dir*)
Bases: object

Manages the paths to the different files and directories of the application.

The EMSM distinguishes two primary folders: The *instance* folder, where the worlds, server, configuration and plugins of the user are placed. The *instance* folder can actually be considered to be the working directory of the EMSM. On the other side is the EMSM installation directory. The *emsm* directory. This directory is usually placed in Python's third party library folder (site-packages) and contains the EMSM core application and the core plugins.

1.*emsm* directory:

```
|- emsm
    |- core
        |- lib
            |- ...
        |- __init__.py
        |- application.py
        |- argparse_.py
        |- base_plugin.py
        |- ...
    |- plugins
        |- __init__.py
        |- backups.py
        |- emsm.py
        |- guard.py
        |- ...
    |- __init__.py
```

2.*instance* folder:

```
|- instance             # usually /opt/minecraft
    |- conf             # the emsm configuration files
        |- main.conf
        |- server.conf
        |- worlds.conf
    |- plugins          # the user's plugins.
        |- __init__.py
        |- myawesomeplugin.py
        |- ...
    |- plugins_data     # data generated or needed by the plugins
        |- backups
        |- emsm
        |- guard
        |- ...
        |- myawesomeplugin
        |- ...
    |- server           # the server executables
        |- minecraft_server.jar
        |- craftbukkit.jar
        |- ...
    |- worlds           # the data of the worlds (minecraft map, ...)
        |- foo
            |- server.properties
            |- ...
        |- bar
            |- server.properties
            |- ...
    |- logs
        |- emsm.log
        |- emsm.log.1
        |- ...
    |- minecraft.py
```

conf()
Contains the configuration files of the EMSM. **Not** the configuration for the minecraft worlds. These are still located in the world folder.

The directory contains the *main.conf*, *server.conf* and *worlds.conf* file.

The directory is located in the *instance* folder.

create()
Creates the folders used by the EMSM Application.

This method should only be called, after the EMSM downgraded it's priviliges.

emsm()
Returns the path to the *emsm* installation directory. This folder is usually located in Python's `site-packages` directory.

emsm_core()
Returns the path to the *emsm.core* directory.

emsm_plugins()
Returns the path to the *emsm.plugins* directory. The directory contains the core plugins like *worlds*, *server*, *backups*, ...

instance()
The *instance* folder contains all data generated by the EMSM application and the minecraft worlds.

logs()
Contains the EMSM log files.

Note, that this is NOT the log directory of the minecraft server.

plugin_data(*plugin_name*)
This directory contains all data of the plugin with the name *plugin_name*.

The directory is a subfolder of *plugins_data()*.

plugins()
Contains all user plugins and plugin packages.

The directory is located in the *instance* folder.

See also:

emsm_plugins()

plugins_data()
The directory that contains the data generated or used by all plugins.

The directory is located in the *instance* folder.

See also:

plugin_data`()

server()
This directory contains all server executables specified in the `server.conf` configuration file.

The directory is located in the *instance* folder.

See also:

server_()

server_(*server_name*)
This directory contains the server software for the server with the name *server_name*.

The directory is located in the *server()* folder.

Todo

•Try to find better names for *server* and *server_*. They are hard to distinguish.

---

world(*world_name*)
This directory contains the data generated by the minecraft server which powers the world *world_name*. It contains among others those files:

- `server.properties`
- `ops.json`
- `whitelist.json`

Furthermore, it is a child of *worlds()*.

worlds()
Contains for each world in `worlds.conf` one folder that contains the data generated by the minecraft server.

The directory is located in the *instance* folder.

See also:

world()

emsm.core.plugins

exception emsm.core.plugins.**PluginException**
Bases: Exception

Base class for all exceptions in this module.

exception emsm.core.plugins.**PluginImplementationError**(*plugin*, *msg*)
Bases: *emsm.core.plugins.PluginException*

Raised if a plugin is not correct implemented.

exception emsm.core.plugins.**PluginOutdatedError**(*plugin*)
Bases: *emsm.core.plugins.PluginException*

Raised if the version of the plugin is not compatible with the EMSM version.

See also:

- http://semver.org/

class emsm.core.plugins.**PluginManager**(*app*)
Bases: object

Loads and manages all plugins.

If you want to write a plugin and search for the docs, take a look at the *hellodolly* plugin.

See also:

- *BasePlugin*

finish()
Calls *finish()* for each loaded plugin.

get_all_plugins()
Returns all currently loaded plugin instances.

See also:

- *get_plugin_names()*
- *get_plugin()*

get_module(*plugin_name*)
Returns the Python module object that contains the plugin with the name *plugin_name* or None if there is no plugin with that name.

get_plugin(*plugin_name*)
Returns the instance of the plugin with the name *plugin_name* that is currently loaded and used by the EMSM.

get_plugin_names()
Returns the names of all loaded plugins.

get_plugin_type(*plugin_name*)
Returns the plugin class for the plugin with the name *plugin_name* or None, if there is no plugin with that name.

import_from_directory(*directory*)
Imports all Python modules in the `directory`.

Files that do not contain a valid EMSM plugin, are ignored. You can check the log files to see which plugins have been ignored.

See also:

- *import_plugin()*

import_plugin(*path*)
Loads the plugin located at *path*.

Note: The *path* is no longer added to `sys.path` (EMSM Vers. >= 3).

Raises

- ***PluginOutdatedError*** – when the plugin is outdated.
- ***PluginImplementationError*** – when the plugin is not correct implemented.

See also:

- `_plugin_is_outdated()`

init_plugins()
Creates a plugin instance for each loaded plugin class.

When you call this method multiple times, only plugins that have not been initialised already, will be initialised.

plugin_is_available(*plugin_name*)
Returns `True`, if the plugin with the name *plugin_name* is available.

remove_plugin(*plugin_name*, *call_finish=False*)
Unloads the plugin with the name *plugin_name*.

Parameters

- **plugin_name** (*str*) – The name of the plugin that should be unloaded.
- **call_finish** (*bool*) – If true, the *finish()* method of the plugin is called, before it is unloaded.

run()
Calls *run()* of the plugin that has been selected by the command line arguments.

See also:

- *emsm.core.argparse_.ArgumentParser.args()*

setup()
Imports all plugins from the application's plugin directory.

See also:

- *emsm.core.paths.Pathsystem.plugins()*
- *emsm.core.paths.Pathsystem.emsm_plugins()*

emsm.core.server

exception emsm.core.server.**ServerError**
Bases: Exception

Base class for all exceptions in this module.

exception emsm.core.server.**ServerInstallationFailure**(*server*, *msg=None*)
Bases: *emsm.core.server.ServerError*

Raised if a server installation failed.

exception emsm.core.server.**ServerStatusError**(*server*, *status*, *msg=''*)
Bases: *emsm.core.server.ServerError*

Raised if the server should be online/offline for an action but is offline/online.

exception emsm.core.server.**ServerIsOnlineError**(*server*, *msg=''*)
Bases: *emsm.core.server.ServerStatusError*

Raised if the server is online but should be offline.

exception emsm.core.server.**ServerIsOfflineError**(*server*, *msg=''*)
Bases: *emsm.core.server.ServerStatusError*

Raised if the server is offline but should be online.

class emsm.core.server.**BaseServerWrapper**(*app*)
Bases: object

Wraps a minecraft server (executable), NOT a world.

The BaseServerWrapper is initialized using the options in the server.conf configuration file.

Parameters **app** (emsm.core.application.Application) – The parent EMSM application

conf()
Returns the configuration section in the *server.conf* configuration file.

default_start_cmd()
ABSTRACT

Returns the bash command string, that must be executed, to start the server.

If there are paths in the returned command, they must be absolute.

default_url()
ABSTRACT

The URL where the server executable can be downloaded from.

directory()
Absolute path to the directory which contains all server software.

exe_path()
ABSTRACT

Absolute path to the server executable. This file is usually located in *directory()*.

install()
ABSTRACT

Installs the server by downloading it to server(). If the server is already installed, nothing should happen.

This method is called during the EMSM start phase if *is_installed()* returns `False`.

Raises ***ServerInstallationFailure*** –

- when the installation failed.

is_installed()
`True` if the executable has been downloaded and exists, otherwise `False`.

Per default, this method only checks if the *directory()* is empty or not. It can be *overridden* for a more detailed check.

is_online()
Returns `True` if at least one world is currently running with this server.

log_error_re()
ABSTRACT

Returns a regex, that matches every line with a *severe* (critical) error. A severe error means, that the server does not run correct and needs to be restarted.

log_path(*self*)
ABSTRACT

Returns the path of the server log file of a world.

If a relative path is returned, the base path is the world directory.

log_start_re()
ABSTRACT

Returns a regex, that matches the first line in the log file, after a server restart.

classmethod name()
ABSTRACT

The unique name of the server.

Example:

```
"vanilla 1.8"
```

reinstall()
Tries to reinstall the server. If the reinstallation fails, the old `server()` is restored and everything is like before.

Raises

- ***ServerInstallationFailure*** –
  - when the installation failed.
- ***ServerIsOnlineError*** –
  - when a world powered by this server software is online.

start_cmd(*world=None*)
Returns the value for *start_command* in *conf()* if available and the *default_start_cmd()* otherwise.

Parameters **world** (*str*) – The name of an EMSM world. The start command can be overridden for each world. We will look for a custom start command in the worlds configuration file first.

translate_command(*cmd*)
ABSTRACT

Translates the vanilla server command *cmd* to a command with the same meaning, but which can be understood by the server.

Example:

```
>>> # A BungeeCoord wrapper would do this:
>>> bungeecord.translate_command("stop")
"end"
>>> bungeecord.translate_command("say Hello World!")
"alert Hello World!"
```

url()
Returns the url in *conf()*, if available. Otherwise the value of *default_url()*.

world_address(*world*)
ABSTRACT

Returns the address (ip, port) which is binded by the world. (None, None) should be returned, if the binding can not be retrieved.

If the server is binded to all ip addresses, return the emtpy string `""` for the ip address.

The port should be returned as integer. If it can not be retrieved, return None.

class emsm.core.server.**ServerManager**(*app*)
Bases: object

Manages all server wrappers, owned by an EMSM application.

The ServerManager helps to avoid double instances of the same server wrapper.

add(*server_class*)
Makes the *server_class* visible to this manager. The class must implement all abstract methods of *BaseServerWrapper* or unexpected errors may occure.

Raises

- **TypeError** – if *server_class* does not inherit *BaseServerWrapper*
- **ValueError** – if another wrapper with the *name()* of *server_class* has already been registered.

get(*servername*)
Returns the ServerWrapper with the name *servername* and None, if there is not such a server.

get_all()
Returns a list with all loaded ServerWrapper.

get_by_pred(*pred=None*)
Almost equal to:

```
>>> filter(pred, ServerManager.get_all())
...
```

get_names()
Returns a list with the names of all server.

get_selected()
Returns all server that have been selected per command line argument.

emsm.core.version

emsm.core.version.**VERSION = '5.0.6b0'**
The version of the EMSM.

Take a look at http://semver.org for more information.

emsm.core.worlds

exception emsm.core.worlds.**WorldError**
Bases: Exception

Base class for all other exceptions in this module.

exception emsm.core.worlds.**WorldStatusError**(*world*, *is_online*)
Bases: *emsm.core.worlds.WorldError*

Raised, if a task can not be done because of the current status of the world (online or not online).

exception emsm.core.worlds.**WorldIsOnlineError**(*world*)
Bases: *emsm.core.worlds.WorldStatusError*

Raised if a world is online but should be offline.

exception emsm.core.worlds.**WorldIsOfflineError**(*world*)
Bases: *emsm.core.worlds.WorldStatusError*

Raised if a world is offline but should be online.

exception emsm.core.worlds.**WorldStartFailed**(*world*)
Bases: *emsm.core.worlds.WorldError*

Raised if the world start failed.

exception emsm.core.worlds.**WorldStopFailed**(*world*)
Bases: *emsm.core.worlds.WorldError*

Raised if the world stop failed.

exception emsm.core.worlds.**WorldCommandTimeout**(*world=''*)
Bases: *emsm.core.worlds.WorldError*

Raised, when the server did not react in x seconds.

class emsm.core.worlds.**WorldWrapper**(*app*, *name*)
Bases: object

Provides methods to handle a minecraft world like *start()*, *stop()*, *restart()*, ...

The WorldWrapper is initialised using the configuration options in the section with the name *name* in the server.conf configuration file.

address()
Returns the binding (ip, port) of the world. If those values can not be retrieved, (None, None) is returned.

conf()
The configuration section of this world in the name.world.conf configuration file:

```
.. code-block:: ini
```

morpheus.world.conf [world] server = vanilla 1.8

See also:

- *WorldConfiguration*

directory()
Returns the directory that contains all world data generated by the minecraft server.

If the world is run by the *mojang* minecraft server, this directory contains the `server.properties`, `whitelist.json`, ... files.

install()
Creates the directory of the world.

See also:

- meth:*directory*

is_installed()
Returns `True` if the *directory()* of the world exists, otherwise `False`.

See also:

- *directory()*
- *install()*
- *uninstall()*

is_offline()
Returns `True` if the world is currently **not** running.

is_online()
Returns `True` if the world is currently running.

kill_processes()
Kills all processes with a pid in *pids()*.

Signals:

- *world_about_to_stop*
- *world_stopped*
- *world_stop_failed*

Raises ***WorldStopFailed*** – if the process could not be killed.

See also:

- *pids()*

latest_log()
Returns the log of the world since the last start. If the logfile does not exist, an empty string will be returned.

name()
The name of the world.

This is the name of the configuration section in `worlds.conf` and the folder name in the `worlds` directory.

open_console()
Opens **all** screen sessions whichs pid is in *pids()*.

Raises ***WorldIsOfflineError*** – if the world is offline.

pids()
Returns a list with the pids of the screen sessions with the name *screen_name()*.

restart(*force_restart=False*, *stop_args=None*)
Restarts the server.

Parameters

- **force_restart** (*bool*) – Forces the stop of the server by calling `kill_processes`() if necessairy.
- **stop_args** (*dict*) – If provided, these values are passed to *stop()*.

Signals:

- *world_about_to_stop*
- *world_stopped*
- *world_stop_failed*
- *world_about_to_start*
- *world_started*
- *world_start_failed*

Raises

- ***WorldStopFailed*** – if the world could not be stopped.
- ***WorldStartFailed*** – if the world could not be restarted.

See also:

- *stop()*
- *start()*

screen_name()
Returns the name of the screen sessions that run the server of this world.

send_command(*server_cmd*)
Sends the given command to all screen sessions with the world's screen name.

Raises ***WorldIsOfflineError*** – if the world is offline.

> **Warning:** There is no guarantee, that the server reacted to the command.

send_command_get_output(*server_cmd*, *timeout=10*, *poll_intervall=0.2*)
Like `send_commmand()` but checks every *poll_intervall* seconds, if content has been added to the logfile and returns the change. If no change could be detected after *timeout* seconds, an error will be raised.

Raises

- ***WorldIsOfflineError*** – if the world is offline.
- ***WorldCommandTimeout*** – if the world did not react within *timeout* seconds.

server()
The `ServerWrapper` for the server that runs this world.

set_server(*server*)
Changes the server that runs this world. The world has to be offline.

Parameters **server** (*emsm.core.server.ServerWrapper*) – The new server

:raises WorldIsOnlineError* if the world is online.

start(*wait_check_time=0.1*)
Starts the world if the world is offline. If the world is already online, nothing happens.

Signals:

- *world_about_to_start*
- *world_started*
- *world_start_failed*

Parameters **wait_check_time** (*float*) – Time waited, before checking if the server actually started.

Raises ***WorldStartFailed*** – if the world could not be started.

stop(*force_stop=False*, *message=None*, *delay=None*, *timeout=None*)
Stops the server.

Parameters

- **message** (*str*) – Send to the world before the *stop()* command is executed.
- **delay** (*float*) – Time in seconds that is waited between seding the *message* and executing the :meth‘stop‘ command.
- **timeout** (*float*) – Maximum time in seconds waited for the server stop after executing the *stop()* command.
- **force_stop** (*bool*) – If true and the server could not be stopped, *kill_processes()* is called.

Signals:

- *world_about_to_stop*
- *world_stopped*
- *world_stop_failed*

Raises ***WorldStopFailed*** – if the world could not be stopped.

See also:

- *kill_processes()*
- *is_offline()*

uninstall()
Stops the world and removes the world directory.

See also:

- *kill_processes()*
- *directory()*

world_about_to_start = <blinker.base.NamedSignal object at 0x7fe9e9d6e208; 'world_about_to_start'>
Signal, that is emitted when a world is about to start.

world_about_to_stop = <blinker.base.NamedSignal object at 0x7fe9e9d6e2b0; 'world_about_to_stop'>
Signal, that is emitted when a world is about to be stopped.

world_start_failed = <blinker.base.NamedSignal object at 0x7fe9e9d6e278; 'world_start_failed'>
Signal, that is emitted when a world could not be started.

world_started = <blinker.base.NamedSignal object at 0x7fe9e9d6e240; 'world_started'>
Signal, that is emitted when a world has been started.

world_stop_failed = <blinker.base.NamedSignal object at 0x7fe9e9d6e320; 'world_stop_failed'>
Signal, that is emitted when a world could not be stopped.

world_stopped = <blinker.base.NamedSignal object at 0x7fe9e9d6e2e8; 'world_stopped'>
Signal, that is emitted when a world has been stopped.

world_uninstalled = <blinker.base.NamedSignal object at 0x7fe9e9d6e160; 'world_uninstalled'>
Signal, that is emitted when a world has been uninstalled.

worldpath_to_ospath(*rel_path*)
Converts *rel_path*, that is relative to the root directory of the minecraft world, into the absolute path of the operating system.

Example:

```
>>> # I assume the EMSM root is: "/home/minecraft"
>>> foo.name()
"foo"
>>> foo.worldpath_to_ospath("server.properties")
"/opt/minecraft/worlds/foo/server.properties"
```

See also:

- *directory()*

class emsm.core.worlds.**WorldManager**(*app*)
Bases: object

Works as a container for the *WorldWrapper* instances.

get(*worldname*)
Returns the *WorldWrapper* for the world with the name *worldname* or None if there is no world with that name.

get_all()
Returns a list with all loaded worlds.

get_by_pred(*pred=None*)
Filters the worlds using the predicate *pred*.

Example:

```
>>> # All running worlds
>>> wm.get_by_pred(lambda w: w.is_online())
...
```

See also:

- *get_all()*

get_names()
Returns a list with the names of all worlds.

get_selected()
Returns all worlds that have been selected per command line argument.

See also:

- *emsm.core.argparse_.ArgumentParser.args()*

load_worlds()
Loads all worlds declared in the `worlds.conf` configuration file.

See also:

- `WorldsConfiguration`

If you want to know, how the EMSM works, you are probably faster by reading the source code than this API documentation. The code is written to be read by other persons and quite easy to understand. Since the EMSM does not use threads, you can simply follow the function calls, starting in `__init__.py`. I guess it won't last longer than **1.5 hours** to read and understand how the EMSM works.

About the depencies

The EMSM depends on some tools and Python packages.

One of them (the most important) is screen, which is used to run the minecraft worlds in the background.

We also depend on some Python packages, which are all available via PyPi.

CHAPTER 4

Changelog

This log contains only the changes beginning with version *3.1.1-beta*.

- 5.0.0-beta
  - The `worlds.conf` configuration file has been replaced with a configuration file for each world.

    Upgrading is easy: For each world in `worlds.conf`, create a configuration file `name.world.conf` in the configuration directory:

    The *morpheus* section in `worlds.conf`:

    ```
    [morpheus]
    server = vanilla 1.11
    enable_initd = yes
    stop_timeout = 10
    ```

    becomes the `morpheus.world.conf` configuration file, with the content:

    ```
    [world]
    server = vanilla 1.11
    stop_timeout = 10

    [plugin:initd]
    enable = yes
    ```

  - Custom plugins still work, if you update the `VERSION` attribute.
  - **changed** The *enable_initd* option has been replaced with a new option *enable* in the `plugin:initd` configuration section (checkout the documentation of the *initd* plugin for more information).
  - **added** You can now overridde the server *start_command* for each world.
  - **added** The *backups* plugin has now an *exclude* options, which allows you to exclude world directories from the backup. (issue #58)
  - **added** Some *backups* options can be overridden for each world.
  - **added** *emsm.core.base_plugin.BasePlugin.world_conf()*

  - **added** *emsm.core.base_plugin.BasePlugin.global_conf()*
  - **deprecated** *emsm.core.base_plugin.BasePlugin.conf()*, use `global_conf()` instead.
- 4.0.13-beta
  - **fixed** The start command option `nogui` of the forge server
- 4.0.12-beta
  - **fixed** issue #35
  - **fixed** The start command option `nogui` of the vanilla server
- 4.0.5-beta
  - The server executables are now always placed in a subdirectory of `INSTANCE_ROOT/server/`.
  - **removed** `emsm.core.server.BaseServerWrapper.server()`
  - **added** *emsm.core.server.BaseServerWrapper.directory()*
  - **added** *emsm.core.server.BaseServerWrapper.exe_path()*
  - The *start_command* in the `server.conf` accepts due to the changes above now these placeholders:
    - `{server_exe}` Points to the server executable
    - `{server_dir}` Points to the directory which contains all server software.
  - *added* *emsm.core.paths.Pathsystem.server_()*
- 4.0.0-beta
  - **changed** The EMSM is now a valid Python package available via PyPi.
  - **cleaned** the documentation
  - EMSM upgrade from version 3 beta:
    1. Install the EMSM package

```
$ sudo pip3 install emsm
```

    2. Remove obsolete folders and files:

```
$ rm README.md
$ rm LICENSE.md
$ rm minecraft.py
$ rm .gitignore

$ rm -rf .git/
$ rm -rf docs/
$ rm -rf emsm

# You probably want to keep your own plugins. So modify the
# command to delete only the EMSM plugins (worlds, server, ...).
$ rm -r plugins/*
```

    3. Create the `minecraft.py` file:

```
#!/usr/bin/env python3

import emsm
```

```
# Make sure, the instance folder is correct.
emsm.run(instance_dir = "/opt/minecraft")
```

```
$ chmod +x /opt/minecraft/minecraft.py
$ chown minecraft:minecraft /opt/minecraft/minecraft.py
```

- 3.1.1-beta
  - **added** *emsm.core.server.BaseServerWrapper.world_address()* method
  - **added** *emsm.core.server.BaseServerWrapper.log_error_re()* method
  - **added** *termcolor* as Python requirement
  - **added** *colorama* as Python requirement
  - **added** *pyyaml* as Python requirement
  - **added** *wait_check_time* parameter to *emsm.core.worlds.WorldWrapper.start()*
  - **updated** the console output: the output is now sorted, colored and consistent
  - **updated** *emsm.plugins.guard* plugin (big rework, take a look)

CHAPTER 5

Contribute

This project needs your help! Please help to improve this application and fork the repository on GitHub.

Bug reports

When you found a bug, please create a bug report.

If you know how to fix the bug, you're welcome to send a *pull request.*

Code

If you like the EMSM and want to contribute to the code, then do it :)

Note, that commits should never go directly to the *master* branch.

Plugins

You wrote a new plugin and want to share it? Great! Write me about it on GitHub and I will add it to the plugins list.

To simplify the usage by other users, you could prepare your plugin:

1. Choose a short and unique name for your plugin.
2. Create a `plugin package`, that contains the source file and data, which comes with your plugin.

3. Add a small reST docstring to your plugin. If you don't know how to do this, you can take a look at the source code of some other plugins. It's quite easy.

   The documentation should contain at least those sections:

   - About (What does your plugin?)
   - Download URL
   - Configuration
   - Arguments

Spelling Mistakes

I guess the source code and this documentation contain a lot of spelling mistakes. Please help to reduce them.

CHAPTER 6

License

Source Code

Beginning with version *2.0.1-beta*, the EMSM is published under the MIT license:

```
The MIT License (MIT)

Copyright (c) 2014-2016 Benedikt Schmitt <benedikt@benediktschmitt.de>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Earlier versions have been released under the GNU GPLv3.

Documentation

The documentation is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

CHAPTER 7

About

- Source code: https://github.com/benediktschmitt/emsm

Sources

- Octocat: https://github.com/logos

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

CHAPTER 9

What is the **EMSM** ?

The **Extendable Minecraft Server Manager** (EMSM) is a minecraft server wrapper, that is able to handle multiple minecraft worlds and server versions.

The **EMSM** itself provides only a simple but sufficient **API** to manage the worlds. The rest of the work is done by the plugins. This makes the application easy to extend and maintain.

And the best thing: We support **many server**:

- **vanilla** (mojang server) starting with version 1.2
- **bungeecord**
- **minecraft forge**
- **spigot**

CHAPTER 10

Why should you use the **EMSM**?

- **Python powered**

  Small and readable code base, therefore easy to maintain.
- **Open Source**

  Licensed under the *MIT License*.
- **Portable**

  Needs only Python, screen and java to run and should work on all Linux systems.
- **Cron-Save**

  The EMSM makes sure, that only one instance of the application runs to the same time.
- **InitD**

  Use the `initd` plugin to benefit from the *init.d* service.
- **Simple Configuration**

  Only three simple configuration files with the simple *.ini* syntax.
- **Backup ready**

  Create and manage multiple versions of your worlds with the `backup manager`.
- **Multiple worlds and server**

  This application has been written to administrate and run multiple worlds and server versions to the same time.
- **Beautiful output**

  The EMSM output is colored, so that you only need one view to get the most important information.
- **Guarded worlds**

  The `guard` helps you to monitor the worlds and to react on server issues automatically.
- **Fast learning curve**

  Use the `--help` or `--long-help` argument if you don't know how to use a plugin.

- **Online Documentation**

  You don't come to grips with the configuration? Take a look at this online documentation.

- **Easy to extend**

  Extend the EMSM with a simple plugin and benefit from Pythons great standard library.

CHAPTER 11

Collaboration

Fork this project on GitHub.

Python Module Index

e

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W